Seeking Practical CDCL Insights from Theoretical SAT Benchmarks

Jan Elffers[1], **Jesús Giráldez Cru**[1], **Stephan Gocht**[1], **Jakob Nordström**[1] and **Laurent Simon**[2]
[1] KTH Royal Institute of Technology
[2] Université de Bordeaux
{elffers,giraldez,gocht,jakobn}@kth.se
lsimon@labri.fr

Abstract

Over the last decades Boolean satisfiability (SAT) solvers based on conflict-driven clause learning (CDCL) have developed to the point where they can handle formulas with millions of variables. Yet a deeper understanding of how these solvers can be so successful has remained elusive. In this work we shed light on CDCL performance by using theoretical benchmarks, which have the attractive features of being a) scalable, b) extremal with respect to different proof search parameters, and c) theoretically *easy* in the sense of having short proofs in the resolution proof system underlying CDCL. This allows for a systematic study of solver heuristics and how efficiently they search for proofs. We report results from extensive experiments on a wide range of benchmarks. Our findings include several examples where theory predicts and explains CDCL behaviour, but also raise a number of intriguing questions for further study.

1 Introduction

The emergence of *conflict-driven clause learning (CDCL)* SAT solvers is one of the most impressive success stories of computer science, but also one of its most intriguing mysteries. Though modern CDCL solvers are routinely used to solve real-world instances with hundreds of thousands or even millions of variables, it is still largely unclear how they can achieve this feat. The basic difference between CDCL and classic DPLL backtrack search [Davis *et al.*, 1962] is in how conflicts guide the search for a satisfying assignment. State-of-the-art CDCL solvers employ conflict-directed learning, first introduced in GRASP [Marques-Silva and Sakallah, 1999],[1] and conflict-directed branching, pioneered by Chaff [Moskewicz *et al.*, 2001], and as reported in [Katebi *et al.*, 2011] these two mechanisms account for most of the performance gain of CDCL over DPLL. Further improvements have been obtained through highly optimized implementations of the basic CDCL algorithm as well as through the use of dozens of sophisticated heuristics.

[1] A similar idea in the context of constraint satisfaction problems (CSPs) was developed in [Bayardo Jr. and Schrag, 1997].

Unfortunately, many CDCL heuristics interact in subtle and unexpected ways, which makes it challenging to assess their relative importance. A natural approach to gain a better understanding would be to collect real-world benchmarks and run experiments with different parameter settings to study how they contribute to overall performance. It seems hard to implement this idea in a satisfactory way, however. The set of available instances is quite limited, and is also a very diverse collection with starkly different properties. Because of this it is hard to obtain statistically significant data which would admit drawing general conclusions.

We propose that a deeper understanding of CDCL solvers can be obtained by subjecting them to carefully chosen theoretical benchmarks with well-studied properties. By instrumenting a solver with "knobs" to tune the settings of various parameters, we want to shed light on what impact each heuristic has on performance and how this correlates with the theoretical properties of the formulas. At first blush it might not be clear why such a study of crafted benchmarks would have any practical relevance, but we consider this to be a worthwhile endeavour for, among others, the following reasons:

- The benchmarks are *scalable*, meaning that one can generate "the same" formula for different sizes and study how performance scales as the instance size increases, rather than just obtaining isolated data points.
- The benchmarks are designed to be *extremal* with respect to different complexity-theoretic properties, meaning that they can be viewed as challenging benchmarks for different heuristics such as, e.g., branching, clause database management, and restart policy.
- Finally, in contrast to most combinatorial benchmarks used in the SAT competitions, which are chosen to be as hard as possible, our benchmarks are constructed so as to be *easy* in the sense of having very short resolution proofs of unsatisfiability that CDCL solvers can potentially find. This means that actual CDCL performance measures the quality of the proof search.

Brief Discussion of Results Let us list some of our conclusions, which we reach by juxtaposing empirical results with theoretical properties of the benchmarks:

1. While the mathematical question of whether restarts are just a helpful heuristic or are fundamentally needed for

CDCL solvers to harness the full power of resolution remains wide open, our results provide empirical evidence that the latter might be the case. Also, adaptive restarts as in [Audemard and Simon, 2012] often work markedly better than the fixed-interval Luby restarts in [Eén and Sörensson, 2004]. However, for some benchmarks it can happen that adaptive restarts are completely switched off, causing terrible performance. This suggests it might be worth considering a combination of dynamic and fixed restarts.

2. Learned clauses are absolutely critical for performance. While the information gathered during conflict analysis is important for guiding other heuristics, the solvers crucially need the exponential increase in reasoning power afforded by also storing the learned clauses to go from tree-like (DPLL-style) to general resolution proofs.
3. For formulas inspired by theoretical time-space tradeoff results, too aggressive clause erasure can incur a stiff penalty in running time also in practice. And when memory is tight, the LBD (literal block distance) heuristic [Audemard and Simon, 2009] often does a particularly good job at identifying useful clauses.
4. For VSIDS variable decisions [Moskewicz *et al.*, 2001] the choice of decay factor can sometimes be vitally important. While we are not sure why, we hypothesize that this might be connected to whether the proof search needs to find DAG-like proofs or whether tree-like proofs are good enough. We can also see that VSIDS decisions can go badly wrong for easy but tricky formulas, which suggests that there is room for further improvements in variable selection heuristics. Somewhat disappointingly, though, we cannot see any support for the hypothesis that the new learning-rate based heuristic [Liang *et al.*, 2016] would be better than VSIDS.

Related Work and Comparison We believe our paper is the very first to implement a comprehensive program as outlined above, but many related ideas can be found in previous works as discussed next. We emphasize that our treatment is very condensed due to space constraints.

Instrumenting CDCL solvers to study the effect of different options has been done in [Lynce and Marques-Silva, 2002; Katebi *et al.*, 2011], and there are also in-depth studies focusing specifically on, e.g., decision heuristics [Biere and Fröhlich, 2015b] and restart schemes [Huang, 2007; Biere and Fröhlich, 2015a]. It seems fair to say, however, that we support a more extensive combination of settings, and that our total computation time is larger by orders of magnitude. Also, these papers all focus mainly on applied benchmarks as found in the SAT competitions.

Using crafted benchmarks instead is, of course, not a new idea, but seems to have been done mainly due to the paucity of real-world instances, not because of connections between theoretical properties and practical performance. An important exception is [Petke and Jeavons, 2009], but here the instances have concrete structural restrictions that make them amenable to explicit algorithms. In contrast, the only guarantee for our instances is that it should in principle be possible to solve them efficiently because of the existence of short resolution proofs. The formulas have been chosen to be extremal, and thus challenging, in different ways given this restriction of small proof size. To achieve this, it is not sufficient to consult general summaries of the proof complexity literature as in [Lauria *et al.*, 2017]—one also needs to think carefully about how to set specific parameters in order to obtain concrete benchmarks with the desired properties. It should be noted in this context, though, that the price of these tight connections to proof complexity is that we can only consider unsatisfiable instances, which is a limitation of our approach.

The concept of scaling the size of instances is well-known from research on random k-SAT (see, e.g., [Crawford and Auton, 1996; Selman *et al.*, 1992]), but is mainly used to check when algorithms hit the exponential brick wall and die (or to study phase transitions). This is conceptually quite distinct from investigating how algorithms scale on problems that are efficiently solvable in principle. Examples of papers that do the latter, in addition to [Petke and Jeavons, 2009], are [Järvisalo *et al.*, 2012; Mikša and Nordström, 2014], but they all use off-the-shelf solvers and not instrumented versions designed to compare different settings as we do.

Outline of This Paper We start with a brief overview of proof complexity in Section 2, giving context for the benchmarks introduced in Section 3. We describe the instrumented CDCL solver in Section 4. Sections 5 and 6 discuss our methodology and present our findings, and we conclude in Section 7 by outlining some possible directions for future research. Our experimental data can be examined at `www.csc.kth.se/~jakobn/CDCL-insights`, where we have also collected benchmarks and solver source code.

2 Proof Complexity

Proof complexity studies how hard it is to prove that formulas in conjunctive normal form (CNF) are unsatisfiable. While the original motivation for this line of research, initiated in [Cook and Reckhow, 1979], was as an approach to prove $\mathsf{P} \neq \mathsf{NP}$, it seems fair to say that most current research in proof complexity is driven by other concerns.

One such concern is the connection to SAT solving. Any SAT algorithm defines a proof system in the sense that the execution trace for an unsatisfiable formula constitutes a polynomial-time verifiable proof of unsatisfiability (also referred to as a *refutation*). Hence, upper and lower bounds for these proof systems provide information about the potential and limitations of the corresponding SAT solvers.

When run on unsatisfiable CNF formulas, CDCL solvers search for proofs in the *resolution* proof system (see, e.g., [Beame *et al.*, 2004]). The most studied complexity measure for resolution is *size*, which gives lower bounds on the running time of CDCL proof search without preprocessing and for which (optimal) exponential lower bounds are known [Urquhart, 1987; Chvátal and Szemerédi, 1988]. Another more recently studied measure is *space*, which corresponds to the number of learned clauses in memory, and for which (again optimal) linear lower bounds have been proven [Alekhnovich *et al.*, 2002; Esteban and Torán, 2001].

For all of these results, the concept of *width*, measured as the size of a largest clause in the proof, has turned out to play a key role. If a formula over n variables has a proof in width w, then this proof has size at most $n^{O(w)}$ by a simple counting argument. Less obviously, strong enough width lower bounds also imply strong lower bounds on proof size [Ben-Sasson and Wigderson, 2001] as well as on space [Atserias and Dalmau, 2008]. The relationships and trade-offs between width and space in resolution are by now fairly well-understood [Ben-Sasson and Nordström, 2008; Ben-Sasson, 2009], as are those between size and space [Ben-Sasson and Nordström, 2008; 2011; Beame *et al.*, 2016; Beck *et al.*, 2013] and size and width [Thapen, 2016].

The natural question how efficiently the CDCL algorithm can search for resolution proofs turns out to be very challenging to answer. [Pipatsrisawat and Darwiche, 2011] showed that CDCL viewed as a proof system is always within a polynomial factor of the best resolution proof. This is not a constructive result, however—it crucially assumes that the solver magically can make the right variable decisions. Also, the solver must never forget even a single learned clause, which is very different from the aggressive erasure used in practice. These restrictions are probably inherent, however, since an actual algorithm would violate widely believed complexity-theoretic assumptions [Alekhnovich and Razborov, 2008].

Intriguingly, the result in [Pipatsrisawat and Darwiche, 2011] crucially relies on the solver making frequent restarts, which is a poorly understood aspect of CDCL. Empirically, it is clear that restarts are very important for performance, but it remains open whether they actually affect the reasoning power of the CDCL method or not. In this context it is natural to consider *regular resolution*, where the proofs—when represented as directed acyclic graphs (DAGs)—satisfy that any variable is resolved at most once along any path in the proof DAG. Regular resolution is strong enough to capture the DP variable elimination algorithm [Davis and Putnam, 1960], but is exponentially weaker than general resolution [Alekhnovich *et al.*, 2007]. Although CDCL without restarts is *not* the same as regular resolution [Beame *et al.*, 2004], the two systems seem "morally close" in that CDCL conflict analysis is regular with respect to the clauses currently in the database. It is therefore natural to ask whether the formulas that separate general resolution from regular resolution can also be used to show that CDCL without restarts cannot simulate the full power of resolution, but attempts to do so have failed [Bonet *et al.*, 2014; Buss and Kołodziejczyk, 2014].

By necessity, our treatment above is very brief and selective—for more details, see, e.g., [Nordström, 2015].

3 Overview of Benchmarks

We consider 9 families of benchmarks constructed from the formulas with asymptotically proven extremal properties discussed in Section 2. Sometimes these formulas have then to be scaled down to obtain reasonably-sized instances for practical experiments, at the price of losing their asymptotically guaranteed theoretical properties, but it seems they keep enough of their character to remain challenging and interesting benchmarks. We stress that all instances have short resolution proofs that can in principle be found by CDCL without any preprocessing, and for most of the formulas even without any restarts given an appropriate (fixed) variable order. It can therefore be argued that CDCL performance on these benchmarks really measures the quality of the proof search.

Tseitin formulas encode systems of linear equations $\bmod 2$ (i.e., XOR constraints) generated from graphs. For long, narrow grids these formulas exhibit asymptotically very strong size-space trade-offs for resolution [Beame *et al.*, 2016; Beck *et al.*, 2013], and we study if scaled-down versions (without these guarantees) exhibit time-space trade-offs in practice and differentiate between different clause database management heuristics. A second family of formulas of seemingly similar flavour are **even colouring formulas** [Markström, 2006] over the same kind of grid graphs.

Pebbling formulas on DAGs always have short proofs, but for appropriately chosen DAGs they have high space complexity [Ben-Sasson and Nordström, 2008] and are also exponentially hard for DPLL [Ben-Sasson *et al.*, 2004]. A closely related family are so-called **stone formulas**, which are easy for general resolution but exponentially hard for regular resolution [Alekhnovich *et al.*, 2007]. Since stone formulas are candidates for showing that CDCL without restarts is weaker than full resolution we are interested in studying how performance on these formulas correlates with restart frequency.

Ordering principle formulas, which claim the existence of finite ordered sets without minimal elements, were shown to have linear-size proofs by [Stålmarck, 1996] (contrary to belief at the time). When converted to 3-CNF these formulas are extremal in that they require proofs with maximally large clauses among all formulas with short proofs [Bonet and Galesi, 2001], and they are exponentially hard for DPLL.

Subset cardinality formulas encode collections of cardinality constraints claiming that both true and false variables are in a strict majority. Randomly structured constraints yield exponentially hard formulas [Mikša and Nordström, 2014], but we focus on "fixed bandwidth" patterns for which the formulas are easy in theory [Van Gelder and Spence, 2010], and even have polynomial-size DPLL proofs.

Clique formulas encode the claim that a graph G on n vertices has a k-clique. In the worst case such formulas are believed to require proofs of size n^k, but we consider complete $(k-1)$-partite graphs, for which the formulas are easily proven unsatisfiable even in regular resolution, although the smallest DPLL proofs scale like n^k [Beyersdorff *et al.*, 2013].

Relativized pigeonhole principle (RPHP) formulas claim that k pigeons (for k a small constant) can fly into $k-1$ holes via n "resting places," where n is the parameter used to scale the formulas. These formulas require proofs of size roughly n^k, and such proofs can be found even by DPLL [Atserias *et al.*, 2016]. For suitably chosen constant k this is thus an example of a family of formulas that have proofs of polynomial but superlinear size. We also study **dominating set formulas** with very similar properties.

4 CDCL Solver and Experimental Set-up

To run our experiments we have built an instrumented CDCL solver on top of Glucose [Audemard and Simon, 2009],

where we have added a rich selection of extra options to analyse the interactions between (1) restart policy, (2) branching, (3) clause database management, and (4) clause learning.

Implementing key features of distinct state-of-the-art solvers in such a manner that different options can be combined in a sensible way poses nontrivial challenges and sometimes forces hard choices. We have striven to implement distinctive solver features in a faithful way, and have spent significant effort on verifying the instrumented solver by comparing the "MiniSat-like settings" to original MiniSat, et cetera, checking that performance was as expected.

For **restart policy** we consider adaptive restarts (as in Glucose), Luby-sequence-based restarts (as in MiniSat) with different frequencies, and also restarts turned completely off to study the reasoning power of CDCL without restarts.

For **variable decisions** we explore VSIDS branching with different settings of the decay factor, where a larger decay factor corresponds to remembering more of the conflict history, as well as learning rate-based branching (LRB) as described in [Liang *et al.*, 2016]. We have also run experiments with fixed-order branching (chosen to be good for the benchmark in question when a good fixed order exists) in order to obtain a baseline against which to compare other settings, and have done some limited experiments with random decisions to study when using more sophisticated heuristics is crucial.

For the **phase** we investigate fixed phases (set to 0 or chosen randomly) , standard phase saving as introduced in [Pipatsrisawat and Darwiche, 2007], branching against the phase, and choosing the phase independently at random every time.

To study clause database management we consider different policies for clause erasure (how many learned clauses to erase) and assessment (which learned clauses to erase). We also explore the effect of turning clause learning off, switching between DPLL-style and CDCL-style search, but implementing DPLL in such a way that other heuristics such as VSIDS still make sense. For **clause erasure** we consider MiniSat-style (most aggressive) and Glucose-style (less aggressive),[2] as well as a significantly more generous version of Glucose. As extremal cases we study no erasures and DPLL-style search (essentially erasing all clauses). For **clause assessment** we consider the standard settings in MiniSat and Glucose as well as a totally random heuristic to check how well the state-of-the-art heuristics identify important clauses.

Regarding **learning scheme** it seems that 1UIP clause learning is what is used by all state-of-the-art solvers, but we also investigate last (decision) UIP learning in order to have something with which to compare.

Our focus in this work is strictly on understanding basic CDCL proof search. However, since **preprocessing** is an integral part of successful CDCL implementations we have run experiments both without preprocessing and with standard preprocessing as in MiniSat/Glucose turned on.

We have run our experiments on a cluster with 6 AMD Opteron 6238 (2.6 GHz) cores with 16 GB of memory, using a timeout of 5000 seconds. Ideally, we would have liked to investigate the full Cartesian product of combinations of settings described above. This is simply not feasible, however, due to the combinatorial explosion of cases. Instead, we have selected a generous subset of more than 650 different solver configurations, which still makes it possible to study how most settings interact. Already this more limited set of experiments required hundreds of years of computation time in total, and generated massive amounts of data to analyse.

[2]For MiniSat the initial clause database size depends on the input. Glucose starts with a fixed, potentially smaller, size, but increases it faster so that for large timeouts it will become much larger.

5 Methodology and Critical Discussion

Before presenting our results, let us discuss how we analyse our experimental data. We start by highlighting some caveats.

First and foremost, it is crucial to remember that correlation is not causation. However, an important aspect of our work is that the interpretation of the correlations we see is informed by our theoretical understanding of the benchmarks, and this gives us added confidence when empirical observations match what theory predicts. Still, some caution is in order. As already discussed, to run the experiments the formulas sometimes have to be scaled down to parameter ranges where theorems about asymptotic mathematical properties no longer apply. Also, even when these properties do hold, we have no guarantee that they explain what we see—solver performance might depend on other characteristics of the benchmarks which we fail to consider. However, while these inherent problems are essential to keep in mind, they do not preclude a meaningful, informative analysis. And even for results where no theoretical explanation readily presents itself, our use of extremal formulas has arguably helped to uncover previously unknown phenomena that can stimulate follow-up studies with more fine-grained tools.

A clear advantage of theory benchmarks is that we can solve "the same problem" in different sizes and study how performance scales. This tacitly assumes, however, that it is meaningful also from a solver point of view to consider different formulas generated by varying a parameter as closely related. Most often we can make the notion of sameness formal, in that partial assignments to variables in a larger instance can be applied to yield a residual formula that is a smaller instance in the same family, but this is not always the case.

As already pointed out, a limitation of our set-up is the exclusive focus on unsatisfiable instances. This restriction is dictated by proof complexity, where the relevant concepts are defined only for unsatisfiable formulas, It is also an interesting setting since in many real-world applications proving unsatisfiability is the objective, and conventional wisdom is that such instances are often the hardest ones. Nevertheless, in practice solvers need to perform well also when there exist solutions, and some heuristics are designed to balance performance on both satisfiable and unsatisfiable instances.

Let us now describe the formal setting for our analysis. We can view the instrumented CDCL solver as a black box with knobs $K_i, i \in [L] = \{1, \ldots, L\}$, that can each be turned to one of a fixed number of settings $\mathcal{K}_i$ (where different knobs decide, e.g., restart policy or decision heuristic). A *solver configuration* $C = (k_1, \ldots, k_L)$ is a tuple in the product space $\mathcal{C} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_L$ of possible settings. Given a family of 30–50 instances of the same problem in different sizes, we

want to understand which configurations are best (which we expect will depend on the particular properties of this family).

To measure the performance of a solver configuration C on a benchmark family, we run the solver with timeout T and compute PAR (penalized average runtime) scores with penalty P, i.e., averaging all running times but charging a penalty of $P \cdot T$ for timeouts (where we use $P = 2$, though the exact choice of P does not seem to matter much). We also study other measures such as number of conflicts, but will mostly ignore this in what follows due to space constraints.

Determining which configuration is best is easy—just take the lowest PAR score. But we are interested in the more refined question whether there are certain knobs that are particularly important. This might seem like a standard problem in statistics, but the issue here is that state-of-the-art-solvers are essentially *deterministic* algorithms. This means that, in principle, there is no uncertainty or noise, but we have full information about all data points.[3] We approach this conceptual problem in the following way: Consider a set $S \subseteq [L]$ of $|S| = \ell \ll L$ knobs (we use $\ell \leq 3$) adjusted to settings $k'_i \in \mathcal{K}_i, i \in S$. The value of this *subconfiguration* is the expected PAR score when the rest of the knobs $K_i, i \in [L] \setminus S$, are set at random, i.e., the average score of all configurations $\mathcal{C}' = \{(k_1, \ldots, k_L) \,|\, k_i = k'_i, i \in S\}$. Then a subconfiguration would seem to be good if the average PAR score for $\mathcal{C}'$ is lower than the total average for all configurations $\mathcal{C}$.

But this begs the question—by definition, we should expect the PAR score average for $\mathcal{C}'$ to be better than the global average for $\mathcal{C}$ roughly 50% of the time. How can we decide when the difference is significant? To answer this question we use the standard deviation measure. Suppose that the total number of configurations is $|\mathcal{C}| = N$, so that we have a global average of N PAR scores, and that the number of configurations with $k'_i, i \in S$, fixed is $|\mathcal{C}'| = M \ll N$. What we can do is to sample M out of the N PAR scores completely at random and compute the standard deviation of such an experiment. If the PAR score average of $\mathcal{C}'$ is clearly farther away from the global average than this standard deviation, then this indicates that these particular knobs and settings $k'_i, i \in S$, indeed seem to have a significant influence on performance. A potential source of concern here is that we will measure this for all subconfigurations with ℓ knobs, but this is only a vanishingly small number of values compared to the total number of random samples $\binom{N}{M}$. Also, we are not using this test in isolation to try to compute a formal significance value, but as a way to crunch the data and find subconfigurations that should be subjected to an in-depth analysis by other means.

While this test provides a very good overview it has two disadvantages. Firstly, if one knob is very important, then it can drown out the signals from other knobs. To counter this effect we can restrict the parameter space to the good setting(s) for this first knob and then repeat the test. Secondly, the test is designed to detect when a small number of knobs have significant impact regardless of other knobs, but it will not show if there is a large number of knobs that critically depend on being set to the right values simultaneously.

In order to be able to discover solver configurations with dependencies between a large number of knobs,[4] and also to get a more qualitative analysis, we use *heatmaps*, where we visualize the results for a benchmark family in a coloured matrix with a row for every solver configuration and a column for each benchmark instance. The colour of a cell (C, i) indicates the running time for configuration C when run on instance i. We then sort the rows by colour to get an indication of which combinations of settings perform well or badly for a family. This kind of visual analysis is often a very helpful complement to the more quantitative analysis described above, and can also reveal patterns that would not be possible to observe using just PAR scores. For a further in-depth analysis of some of our findings we also use other tools that we cannot describe here due to space constraints.

To present our results in Section 6 we use plots of running times for different solver configurations as instance size grows, and also boxplots (quartiles with 1.5 IQR whiskers) to visualize the difference when switching between two particular settings for one knob (in these boxplots we count timeouts, corresponding to penalty $P \to \infty$). We want to emphasize, though, that while these plots are useful to illustrate our final findings, this is not how we sift through the vast amounts of data from our experiments to actually discover these findings.

6 Experimental Results and Analysis

In this section we summarize the results of our experiments.

Restarts The role of restarts is one of the most intriguing, and theoretically challenging, problems regarding CDCL solvers. Is restarting just a useful heuristic, or could it be that the addition of frequent restarts actually changes the underlying method of reasoning, making it theoretically stronger?

Although we have only empirical conclusions, our data supports the hypothesis that frequent restarts are important to harness the full power of resolution. Specifically, we can see that frequent restarts are crucial for stone formulas, which are hard for the restricted subsystem regular resolution (although, as discussed previously, in our experiments we cannot quite push parameters to the range where we have theoretical hardness guarantees). Frequent restarts also seem to be important when a good resolution proof works by learning XORs or equivalences of variables—or more complicated functions—in a bottom-up fashion (as also proposed by participants in the Dagstuhl workshop 15171 *Theory and Practice of SAT Solving*). This benefit of restarts can be seen clearly for pebbling formulas (Figure 1).

Adaptive restarts as in Glucose often perform better than static Luby restarts. We can also see, however, that for some benchmarks the adaptive restarts go badly wrong—restarts are completely blocked since the solver thinks it is closing in on a satifying assignment, which it is not, whereas a static Luby restart sequence works much better. This suggests that a further improvement of CDCL restarts could be to trigger

[3] To create a random distribution one could randomize heuristics [Lynce and Marques-Silva, 2002] or shuffle benchmarks [Katebi *et al.*, 2011], but we want to avoid such drastic modifications since they could potentially distort the measurements we want to make.

[4] Jumping a bit ahead, we found no such dependencies, though.

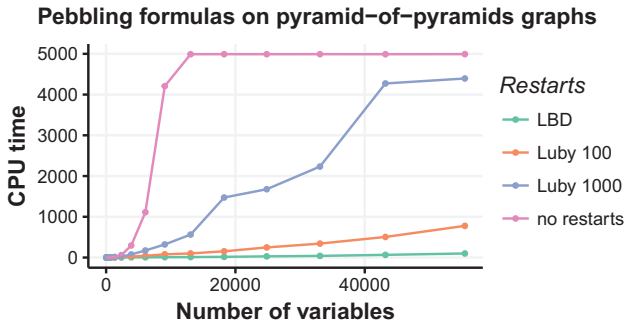

Figure 1: Different restart policies for pebbling formulas (using VSIDS decay factor 0.80 and Glucose defaults for other settings).

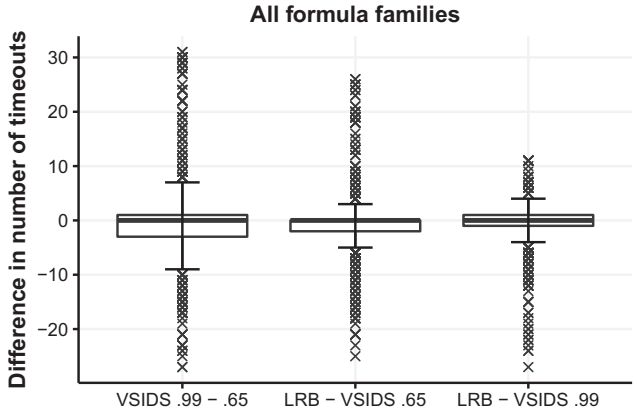

Figure 2: Comparing variable decision strategies for all families.

restarts by a combination of adaptive and Luby heuristics, to make sure that the solver does not have too long sequences of no restarts. (However, for one particular benchmark family, namely clique formulas, it is essential that restarts are switched off completely, but this intriguing finding we are currently unable to explain.)

Variable Decisions One of the decision heuristics we evaluate is LRB [Liang *et al.*, 2016], proposed as an improvement of VSIDS. Our data does not support the hypothesis that LRB would be better. Sometimes it is better; sometimes it is worse; most of the time there is no significant difference from VSIDS with decay factor 0.99 (see the rightmost boxplot in Figure 2, showing that the number of timeouts for each benchmark family for VSIDS with decay factor 0.99 minus the timeouts for LRB is close to 0, though there is a slightly larger amount of outliers in favour of LRB).

Another interesting observation is that there is no single best value for the decay factor for the standard VSIDS heuristic: for some benchmarks it is better to use a low decay factor; for others a high value is better (see the leftmost boxplot in Figure 2). For some families the choice is crucial. For instance, for a high VSIDS decay factor the partial ordering principle formulas become impossible to solve, whereas for a lower factor (corresponding to a higher rate of forgetting the history of conflicts) they are very easy (see Figure 3). In

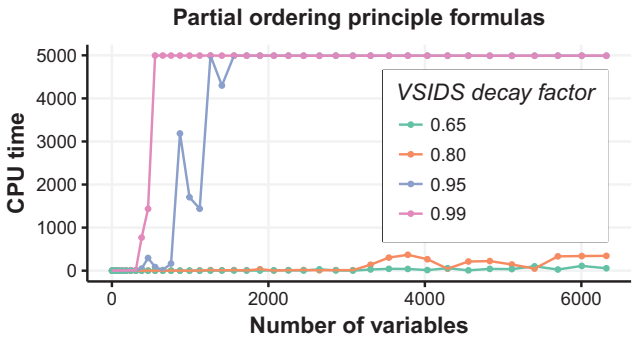

Figure 3: Different VSIDS decay factors for partial ordering principle formulas (using Glucose defaults for other settings).

contrast, for dominating set and RPHP formulas a high decay factor seems better. Intriguingly, for these formulas tree-like resolution proofs are (asymptotically) optimal. These results raise the question whether perhaps having a larger VSIDS decay factor somehow makes the solver proof search closer to tree-like resolution. More experiments on new benchmarks would be needed for a more in-depth investigation of this.

We can also observe that making variable decisions randomly is consistently a terrible option (in contrast to, e.g., random phase or random clause assessment as discussed below, which is usually not good but also not catastrophic).

Phase Saving The influence of phase saving might well be the least well understood aspect of CDCL from a mathematical point of view, with no theoretical studies made as far as we are aware. We observe that standard phase saving as implemented in modern CDCL solvers is often essential, and for some of our benchmarks it is the decisive parameter for good performance. (The one clear exception in our data set are the clique formulas—which also behave strangely with respect to restarts—for which phase saving is bad).

Also, dynamic phases often seem better than fixed phases, mostly because choosing a bad fixed phase can be fatal. In particular, the fixed-0 phase originally used in MiniSat is very bad for several benchmark families, and for some even having no phase (just doing random coin flips every time) can be better than having the wrong phase (see Figure 4 for an example). Note that this is in clear contrast to variable decisions, where making random choices is always a horrible idea.

For benchmarks where frequent restarts are essential, phase saving is also important in that it gives a additional boost. Additional investigations would be needed to clarify the dependencies here, though, since phase saving is already a very influential factor on its own.

Clause Erasure A question that was debated at the abovementioned Dagstuhl workshop is whether clause learning really helps to go beyond tree-like resolution to find DAG-like proofs, or whether its main importance is to let variable branching and other heuristics be guided by the conflict analysis. We can see beyond doubt that for formulas that are hard for tree-like resolution (but easy for general resolution) clause

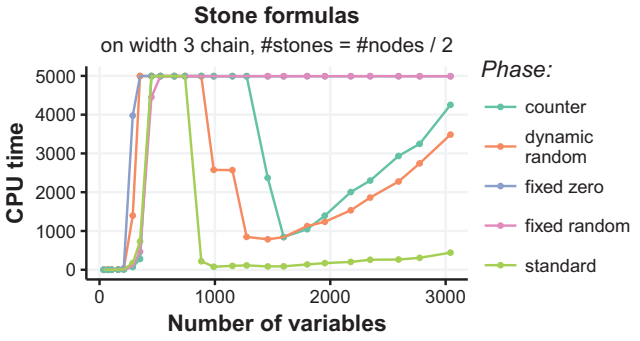

Figure 4: Different phase selection heuristics for stone formulas (using Glucose default settings with VSIDS decay factor 0.95).

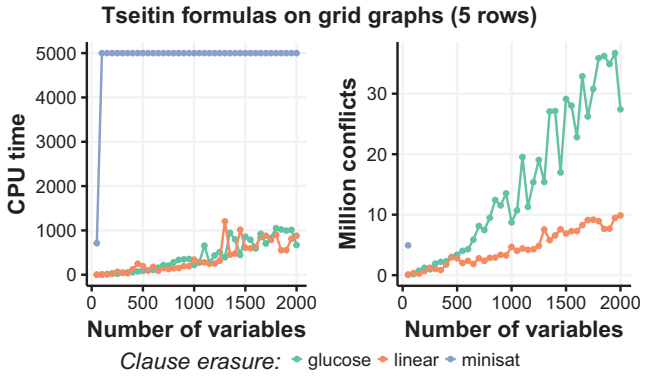

Figure 5: Different clause erasure policies for Tseitin formulas (using Glucose default settings with VSIDS decay factor 0.80).

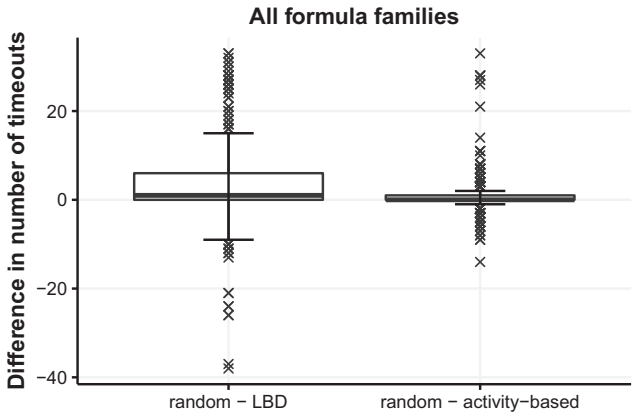

Figure 6: Comparing clause assessment strategies for all families.

learning has a dramatic impact, allowing the solver to go from tree-like to DAG-like proof search. For such formulas the DPLL option in our instrumented solver is disastrous, while it can sometimes perform reasonably well (though still worse than CDCL) for formulas which have short tree-like proofs.

For challenging benchmark families, where solver running times are clearly superlinear in the input size, aggressive clause erasure becomes important. The reason for this seems to be that unit propagation just takes too much time otherwise. However, in terms of the quality of the proof search, measured as the number of conflicts, having more clauses in memory is always more helpful for all of our benchmarks, and if the clause erasure get too aggressive, like in the MiniSat default, then performance can suffer badly. It is not a priori clear that this would always have to be so—one could envision that there would be benchmarks where pruning of the clause database would be useful to get rid of "junk clauses" that give unhelpful unit propagations at the current stage of the proof search, but we have not found such examples.

As an extreme case of the phenomena discussed above, for our benchmarks derived from theoretical time-space trade-off results we can indeed witness such trade-offs also in practice (which to the best of our knowledge is the first time this has been observed). For these formulas having a larger clause database size helps both in terms of running time and number of conflicts, until the database size just gets too large so that running time suffers (but the number of conflicts stays smaller). We illustrate this behaviour for Tseitin formulas in Figure 5. The small database size for MiniSat makes it time out immediately. Glucose (smaller) and linear (larger) database sizes yield similar performance in terms of running time, but the latter seems to achieve much more efficient proof search judging by the number of conflicts.

Clause Assessment Given that too aggressive clause removal can hurt a SAT solver badly, it is an important question whether there are clause assessment heuristics that compensate for this by helping the solver identify important clauses that should be kept. Perhaps such heuristics could be one reason why SAT solvers do so well in practice?

In particular, it is a natural question whether the widely used literal block distance (LBD) measure can be shown to be good at "keeping the right clauses" for formulas where our experiments show that too aggressive an erasure strategy can be detrimental. Our results are consistent with this hypothesis, showing a clear improvement in running time when using LBD for clause assessment. We also see that LBD-based clause assessment can sometimes compensate for other less good parameter choices.

It is important for the LBD-based heuristic, though, that the clause database size is not too small. Glucose adds an extra bump to the size whenever it learns many so-called *glue clauses* with a low LBD score. This extra bump appears to be crucial, since otherwise the clause database can get clogged up by glue clauses (which are never erased by Glucose). Small-database solver configurations without Glucose bumps explain the large negative outliers in the left boxplot for random vs. LBD clause assessment in Figure 6. While this principally raises the question whether there would be potential for further improvements in Glucose by sometimes erasing glue clauses, we can conclude that the fairly aggressive erasure strategy in Glucose combined with LBD-based clause assessment most often work very well in that the clause database gets an extra size increase when needed.

Interestingly, we can see *no really significant difference* between activity-based clause assessment as in MiniSAT and completely random choices of which clauses to keep or throw

away (so it indeed seems that past performance is no guarantee of future results...). While the left boxplot in Figure 6 illustrates that LBD is noticeably better than random (and the difference would be more dramatic if solver configurations with too small database size were removed), the right boxplot shows that random and activity-based assessments are very close (though there is a very small tendency in favour of activity-based as the first quartile is zero).

Learning Scheme As already mentioned, 1UIP learning is what is used in practice, and there is some theoretical justification for this in that the 1UIP clause will provably maximize the length of the backjump after conflict (see, e.g., [Biere *et al.*, 2009, Chapter 4]). For our benchmarks there is often not too much of a difference between 1UIP and last UIP learning—which could be explained by the fact that these are crafted instances—but to the extent that there are differences it is fair to say that 1UIP is clearly better than last UIP.

Preprocessing For almost all of our formulas preprocessing has no discernable impact on performance. For us this is as expected. Preprocessing is especially effective for recovering higher-order structure that was lost when translating the input to CNF, but this scenario does not apply to our crafted instances. Nevertheless, we have run experiments both with and without preprocessing just as a sanity check, because we wanted to make sure that our results were not heavily dependent on such an integral part of modern CDCL solvers.

7 Concluding Remarks

In this work, we run extensive experiments on crafted benchmarks using an instrumented CDCL solver supporting different combinations of options, and use theoretical properties of the benchmarks to shed light on CDCL performance. We see rich opportunities for continued research in this direction, and quickly list some interesting questions below.

Restarts seem crucial to realize the full power of resolution, but is this just about the restart frequency or can the precise timing sometimes be crucial? And would a combination of adaptive and fixed-interval restarts be even better? Sometimes the exact decay factor in VSIDS is critical—can solvers be made to figure this out adaptively? The importance of phase saving is empirically clear but theoretically very poorly understood, and merits further study. Finally, is it always the case that keeping more clauses is good in terms of quality of the proof search, or could one find examples where it is better to remove "junk clauses" that hinder the search?

Acknowledgements

First and foremost, we are deeply indebted to Karem Sakallah for inspiring us to embark on this massive undertaking, and for taking part in the initial stages of this project. Karem's expertise, energy, and persistence was instrumental in getting this project off the ground. We also want to acknowledge the importance of the Banff workshop 14w5101 *Theoretical Foundations of Applied SAT Solving*, the Dagstuhl workshop 15171 *Theory and Practice of SAT Solving*, and the Fields workshop *Theoretical Foundations of SAT Solving*, which brought the authors together and without which this project would not have happened in the first place.

We are grateful to participants of the *Pragmatics of SAT 2016* and *Theoretical Foundations of SAT Solving* workshops for providing useful feedback on presentations as this work progressed. The fourth author would also like to especially thank Armin Biere, Marijn Heule, and Daniel Le Berre for many helpful discussions. We also gratefully acknowledge the many detailed comments from the anonymous reviewers, which helped us to improve this work substantially.

Our experiments were run using computational resources provided by the Swedish National Infrastructure for Computing (SNIC). Most benchmarks were generated using the tool *CNFgen*, for which we are thankful to Massimo Lauria.

The fifth author was partially supported by the ANR-2016 "SATAS" ANR-15-CE40-0017 National French Project. The other authors were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611. The fourth author was also supported by Swedish Research Council grants 621-2012-5645 and 2016-00782.

References

[Alekhnovich and Razborov, 2008] M. Alekhnovich and A. A. Razborov. Resolution is not automatizable unless W[P] is tractable. *SICOMP*, 38(4):1347–1363, 2008.

[Alekhnovich *et al.*, 2002] M. Alekhnovich, E. Ben-Sasson, A. A. Razborov, and A. Wigderson. Space complexity in propositional calculus. *SICOMP*, 31(4):1184–1211, 2002.

[Alekhnovich *et al.*, 2007] M. Alekhnovich, J. Johannsen, T. Pitassi, and A. Urquhart. An exponential separation between regular and general resolution. *Theory Comp.*, 3(5):81–102, 2007.

[Atserias and Dalmau, 2008] A. Atserias and V. Dalmau. A combinatorial characterization of resolution width. *JCSS*, 74(3):323–334, 2008.

[Atserias *et al.*, 2016] A. Atserias, M. Lauria, and J. Nordström. Narrow proofs be maximally long. *ACM ToCL*, 17(3):19:1–19:30, 2016.

[Audemard and Simon, 2009] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proc. 21st Joint Conf. Artif. Intell. (IJCAI'09)*, pp. 399–404, 2009.

[Audemard and Simon, 2012] G. Audemard and L. Simon. Refining restarts strategies for SAT and UNSAT. In *Proc. 18th Conf. Principles Practice Constr. Prog. (CP'12)*, pp. 118–126, 2012.

[Bayardo Jr. and Schrag, 1997] R. J. Bayardo Jr. and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. 14th National Conf. Artif. Intell. (AAAI'97)*, pp. 203–208, 1997.

[Beame *et al.*, 2004] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *JAIR*, 22:319–351, 2004.

[Beame *et al.*, 2016] P. Beame, C. Beck, and R. Impagliazzo. Time-space tradeoffs in resolution: Superpolynomial lower bounds for superlinear space. *SICOMP*, 45(4):1612–1645, 2016.

[Beck *et al.*, 2013] C. Beck, J. Nordström, and B. Tang. Some trade-off results for polynomial calculus. In *Proc. 45th Symp. Theory Comp. (STOC'13)*, pp. 813–822, 2013.

[Ben-Sasson and Nordström, 2008] E. Ben-Sasson and J. Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *Proc. 49th Symp. Found. Comp. Sci. (FOCS'08)*, pp. 709–718, 2008.

[Ben-Sasson and Nordström, 2011] E. Ben-Sasson and J. Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Proc. 2nd Symp. Innovations Comp. Sci. (ICS'11)*, pp. 401–416, 2011.

[Ben-Sasson and Wigderson, 2001] E. Ben-Sasson and A. Wigderson. Short proofs are narrow—resolution made simple. *J. ACM*, 48(2):149–169, 2001.

[Ben-Sasson *et al.*, 2004] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, 2004.

[Ben-Sasson, 2009] E. Ben-Sasson. Size-space tradeoffs for resolution. *SICOMP*, 38(6):2511–2525, 2009.

[Beyersdorff *et al.*, 2013] O. Beyersdorff, N. Galesi, and M. Lauria. Parameterized complexity of DPLL search procedures. *ACM ToCL*, 14(3):20:1–20:21, 2013.

[Biere and Fröhlich, 2015a] A. Biere and A. Fröhlich. Evaluating CDCL restart schemes. In *Proc. Workshop on Pragmatics of SAT (PoS'15)*, 2015.

[Biere and Fröhlich, 2015b] A. Biere and A. Fröhlich. Evaluating CDCL variable scoring schemes. In *Proc. 18th Conf. Theory Appl. Sat. Testing (SAT'15)*, pp. 405–422, 2015.

[Biere *et al.*, 2009] A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. 2009.

[Bonet and Galesi, 2001] M. L. Bonet and N. Galesi. Optimality of size-width tradeoffs for resolution. *Comp. Complex.*, 10(4):261–276, 2001.

[Bonet *et al.*, 2014] M. L. Bonet, S. Buss, and J. Johannsen. Improved separations of regular resolution from clause learning proof systems. *JAIR*, 49:669–703, 2014.

[Buss and Kołodziejczyk, 2014] S. R. Buss and L. Kołodziejczyk. Small stone in pool. *LMCS*, 10(2):16:1–16:22, 2014.

[Chvátal and Szemerédi, 1988] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4):759–768, 1988.

[Cook and Reckhow, 1979] S. A. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Logic*, 44(1):36–50, 1979.

[Crawford and Auton, 1996] J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3-SAT. *Artif. Intell.*, 81(1-2):31–57, 1996.

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Comm. ACM*, 5(7):394–397, 1962.

[Eén and Sörensson, 2004] N. Eén and N. Sörensson. An extensible SAT-solver. In *6th Conf. Theory Appl. Sat. Testing (SAT'03), Selected Revised Papers*, pp. 502–518, 2004.

[Esteban and Torán, 2001] J. L. Esteban and J. Torán. Space bounds for resolution. *Inform. Comp.*, 171(1):84–97, 2001.

[Huang, 2007] J. Huang. The effect of restarts on the efficiency of clause learning. In *Proc. 20th Joint Conf. Artif. Intell. (IJCAI'07)*, pp. 2318–2323, 2007.

[Järvisalo *et al.*, 2012] M. Järvisalo, A. Matsliah, J. Nordström, and S. Živný. Relating proof complexity measures and practical hardness of SAT. In *Proc. 18th Conf. Principles Practice Constr. Prog. (CP'12)*, pp. 316–331, 2012.

[Katebi *et al.*, 2011] H. Katebi, K. A. Sakallah, and J. P. Marques-Silva. Empirical study of the anatomy of modern SAT solvers. In *Proc. 14th Conf. Theory Appl. Sat. Testing (SAT'11)*, pp. 343–356, 2011.

[Lauria *et al.*, 2017] M. Lauria, J. Elffers, J. Nordström, and M. Vinyals. CNFgen: A generator of crafted benchmarks. In *Proc. 20th Conf. Theory Appl. Sat. Testing (SAT'17)*, pp. 464–473, 2017.

[Liang *et al.*, 2016] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proc. 19th Conf. Theory Appl. Sat. Testing (SAT'16)*, pp. 123–140, 2016.

[Lynce and Marques-Silva, 2002] I. Lynce and J. P. Marques-Silva. Building state-of-the-art SAT solvers. In *Proc. 15th European Conf. Artif. Intell. (ECAI'02)*, pp. 166–170. 2002.

[Markström, 2006] K. Markström. Locality and hard SAT-instances. *JSAT*, 2(1-4):221–227, 2006.

[Marques-Silva and Sakallah, 1999] J. P. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Comp.*, 48(5):506–521, 1999.

[Mikša and Nordström, 2014] M. Mikša and J. Nordström. Long proofs of (seemingly) simple formulas. In *Proc. 17th Conf. Theory Appl. Sat. Testing (SAT'14)*, pp. 121–137, 2014.

[Moskewicz *et al.*, 2001] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. 38th Design Automation Conf. (DAC'01)*, pp. 530–535, 2001.

[Nordström, 2015] J. Nordström. On the interplay between proof complexity and SAT solving. *ACM SIGLOG News*, 2(3):19–44, 2015.

[Petke and Jeavons, 2009] J. Petke and P. Jeavons. Tractable benchmarks for constraint programming. Technical Report RR-09-07, Oxford University, 2009.

[Pipatsrisawat and Darwiche, 2007] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proc. 10th Conf. Theory Appl. Sat. Testing (SAT'07)*, pp. 294–299, 2007.

[Pipatsrisawat and Darwiche, 2011] K. Pipatsrisawat and A. Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011.

[Selman *et al.*, 1992] B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. 10th National Conf. Artif. Intell. (AAAI'92)*, pp. 440–446, 1992.

[Stålmarck, 1996] G. Stålmarck. Short resolution proofs for a sequence of tricky formulas. *Acta Inform.*, 33(3):277–280, 1996.

[Thapen, 2016] N. Thapen. A trade-off between length and width in resolution. *Theory Comp.*, 12(5):1–14, 2016.

[Urquhart, 1987] A. Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.

[Van Gelder and Spence, 2010] A. Van Gelder and I. Spence. Zero-one designs produce small hard SAT instances. In *Proc. 13th Conf. Theory Appl. Sat. Testing (SAT'10)*, pp. 388–397, 2010.